

23 | 基础篇：Linux 文件系统是怎么工作的？

2019-01-11 倪朋飞



23 | 基础篇：Linux 文件系统是怎么工作的？

朗读人：冯永吉 14'20" | 13.14M

你好，我是倪朋飞。

通过前面 CPU 和内存模块的学习，我相信，你已经掌握了 CPU 和内存的性能分析以及优化思路。从这一节开始，我们将进入下一个重要模块——文件系统和磁盘的 I/O 性能。

同 CPU、内存一样，磁盘和文件系统的管理，也是操作系统最核心的功能。

- 磁盘为系统提供了最基本的持久化存储。
- 文件系统则在磁盘的基础上，提供了一个用来管理文件的树状结构。

那么，磁盘和文件系统是怎么工作的呢？又有哪些指标可以衡量它们的性能呢？

今天，我就带你先来看看，Linux 文件系统的工作原理。磁盘的工作原理，我们下一节再来学习。

索引节点和目录项

文件系统，本身是对存储设备上的文件，进行组织管理的机制。组织方式不同，就会形成不同的文件系统。

你要记住最重要的一点，在 Linux 中一切皆文件。不仅普通的文件和目录，就连块设备、套接字、管道等，也都要通过统一的文件系统来管理。

为了方便管理，Linux 文件系统为每个文件都分配两个数据结构，索引节点（index node）和目录项（directory entry）。它们主要用来记录文件的元信息和目录结构。

- 索引节点，简称为 inode，用来记录文件的元数据，比如 inode 编号、文件大小、访问权限、修改日期、数据的位置等。索引节点和文件一一对应，它跟文件内容一样，都会被持久化存储到磁盘中。所以记住，索引节点同样占用磁盘空间。
- 目录项，简称为 dentry，用来记录文件的名称、索引节点指针以及与其他目录项的关联关系。多个关联的目录项，就构成了文件系统的目录结构。不过，不同于索引节点，目录项是由内核维护的一个内存数据结构，所以通常也被叫做目录项缓存。

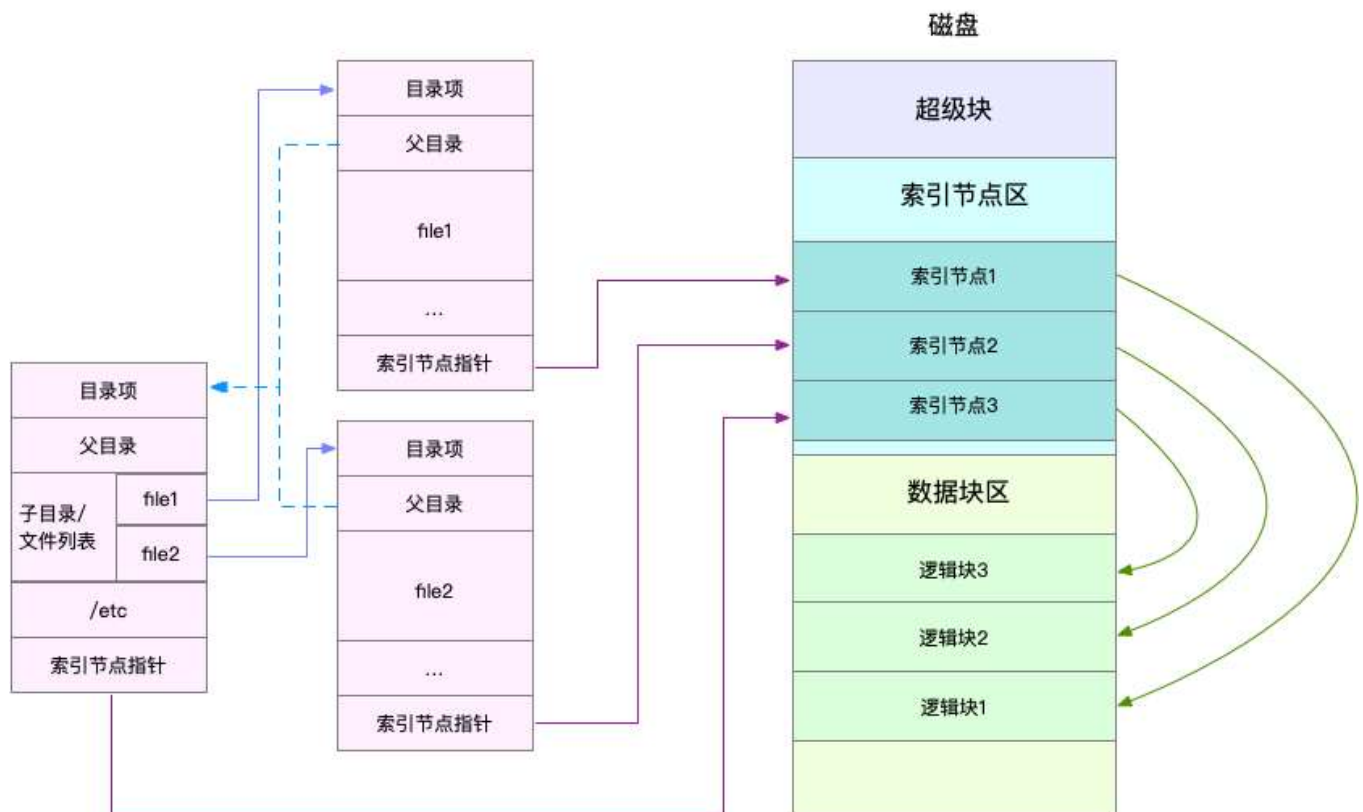
换句话说，索引节点是每个文件的唯一标志，而目录项维护的正是文件系统的树状结构。目录项和索引节点的关系是多对一，你可以简单理解为，一个文件可以有多个别名。

举个例子，通过硬链接为文件创建的别名，就会对应不同的目录项，不过这些目录项本质上还是链接同一个文件，所以，它们的索引节点相同。

索引节点和目录项纪录了文件的元数据，以及文件间的目录关系，那么具体来说，文件数据到底是怎么存储的呢？是不是直接写到磁盘中就好了呢？

实际上，磁盘读写的最小单位是扇区，然而扇区只有 512B 大小，如果每次都读写这么小的单位，效率一定很低。所以，文件系统又把连续的扇区组成了逻辑块，然后每次都以逻辑块为最小单元，来管理数据。常见的逻辑块大小为 4KB，也就是由连续的 8 个扇区组成。

为了帮助你理解目录项、索引节点以及文件数据的关系，我画了一张示意图。你可以对照着这张图，来回忆刚刚讲过的内容，把知识和细节串联起来。



不过，这里有两点需要你注意。

第一，目录项本身就是一个内存缓存，而索引节点则是存储在磁盘中的数据。在前面的 Buffer 和 Cache 原理中，我曾经提到过，为了协调慢速磁盘与快速 CPU 的性能差异，文件内容会缓存到页缓存 Cache 中。

那么，你应该想到，这些索引节点自然也会缓存到内存中，加速文件的访问。

第二，磁盘在执行文件系统格式化时，会被分成三个存储区域，超级块、索引节点区和数据块区。其中，

- 超级块，存储整个文件系统的状态。
- 索引节点区，用来存储索引节点。
- 数据块区，则用来存储文件数据。

虚拟文件系统

目录项、索引节点、逻辑块以及超级块，构成了 Linux 文件系统的四大基本要素。不过，为了支持各种不同的文件系统，Linux 内核在用户进程和文件系统的中间，又引入了一个抽象层，也就是虚拟文件系统 VFS (Virtual File System) 。

VFS 定义了一组所有文件系统都支持的数据结构和标准接口。这样，用户进程和内核中的其他子系统，只需要跟 VFS 提供的统一接口进行交互就可以了，而不需要再关心底层各种文件系统的实现细节。

这里，我画了一张 Linux 文件系统的架构图，帮你更好地理解系统调用、VFS、缓存、文件系统以及块存储之间的关系。



通过这张图，你可以看到，在 VFS 的下方，Linux 支持各种各样的文件系统，如 Ext4、XFS、NFS 等等。按照存储位置的不同，这些文件系统可以分为三类。

- 第一类是基于磁盘的文件系统，也就是把数据直接存储在计算机本地挂载的磁盘上。常见的 Ext4、XFS、OverlayFS 等，都是这类文件系统。
- 第二类是基于内存的文件系统，也就是我们常说的虚拟文件系统。这类文件系统，不需要任何磁盘分配存储空间，但会占用内存。我们经常用到的 /proc 文件系统，其实就是一种最常见的虚拟文件系统。此外，/sys 文件系统也属于这一类，主要向用户空间导出层次化的内核对象。
- 第三类是网络文件系统，也就是用来访问其他计算机数据的文件系统，比如 NFS、SMB、iSCSI 等。

这些文件系统，要先挂载到 VFS 目录树中的某个子目录（称为挂载点），然后才能访问其中的文件。拿第一类，也就是基于磁盘的文件系统为例，在安装系统时，要先挂载一个根目录（/），在根目录下再把其他文件系统（比如其他的磁盘分区、/proc 文件系统、/sys 文件系统、NFS 等）挂载进来。

文件系统 I/O

把文件系统挂载到挂载点后，你就能通过挂载点，再去访问它管理的文件了。VFS 提供了一组标准的文件访问接口。这些接口以系统调用的方式，提供给应用程序使用。

就拿 cat 命令来说，它首先调用 open()，打开一个文件；然后调用 read()，读取文件的内容；最后再调用 write()，把文件内容输出到控制台的标准输出中：

```
1 int open(const char *pathname, int flags, mode_t mode);
2 ssize_t read(int fd, void *buf, size_t count);
3 ssize_t write(int fd, const void *buf, size_t count);
```

 复制代码

文件读写方式的各种差异，导致 I/O 的分类多种多样。最常见的有，缓冲与非缓冲 I/O、直接与非直接 I/O、阻塞与非阻塞 I/O、同步与异步 I/O 等。接下来，我们就详细看这四种分类。

第一种，根据是否利用标准库缓存，可以把文件 I/O 分为缓冲 I/O 与非缓冲 I/O。

- 缓冲 I/O，是指利用标准库缓存来加速文件的访问，而标准库内部再通过系统调度访问文件。
- 非缓冲 I/O，是指直接通过系统调用来访问文件，不再经过标准库缓存。

注意，这里所说的“缓冲”，是指标准库内部实现的缓存。比方说，你可能见到过，很多程序遇到换行时才真正输出，而换行前的内容，其实就被标准库暂时缓存了起来。

无论缓冲 I/O 还是非缓冲 I/O，它们最终还是要经过系统调用来访问文件。而根据上一节内容，我们知道，系统调用后，还会通过页缓存，来减少磁盘的 I/O 操作。

第二，根据是否利用操作系统的页缓存，可以把文件 I/O 分为直接 I/O 与非直接 I/O。

- 直接 I/O，是指跳过操作系统的页缓存，直接跟文件系统交互来访问文件。
- 非直接 I/O 正好相反，文件读写时，先要经过系统的页缓存，然后再由内核或额外的系统调用，真正写入磁盘。

想要实现直接 I/O，需要你在系统调用中，指定 O_DIRECT 标志。如果没有设置过，默认的是非直接 I/O。

不过要注意，直接 I/O、非直接 I/O，本质上还是和文件系统交互。如果是在数据库等场景中，你还会看到，跳过文件系统读写磁盘的情况，也就是我们通常所说的裸 I/O。

第三，根据应用程序是否阻塞自身运行，可以把文件 I/O 分为阻塞 I/O 和非阻塞 I/O：

- 所谓阻塞 I/O，是指应用程序执行 I/O 操作后，如果没有获得响应，就会阻塞当前线程，自然就不能执行其他任务。
- 所谓非阻塞 I/O，是指应用程序执行 I/O 操作后，不会阻塞当前的线程，可以继续执行其他的任务，随后再通过轮询或者事件通知的形式，获取调用的结果。

比方说，访问管道或者网络套接字时，设置 `O_NONBLOCK` 标志，就表示用非阻塞方式访问；而如果不做任何设置，默认的就是阻塞访问。

第四，根据是否等待响应结果，可以把文件 I/O 分为同步和异步 I/O：

- 所谓同步 I/O，是指应用程序执行 I/O 操作后，要一直等到整个 I/O 完成后，才能获得 I/O 响应。
- 所谓异步 I/O，是指应用程序执行 I/O 操作后，不用等待完成和完成后的响应，而是继续执行就可以。等到这次 I/O 完成后，响应会用事件通知的方式，告诉应用程序。

举个例子，在操作文件时，如果你设置了 `O_SYNC` 或者 `O_DSYNC` 标志，就代表同步 I/O。如果设置了 `O_DSYNC`，就要等文件数据写入磁盘后，才能返回；而 `O_SYNC`，则是在 `O_DSYNC` 基础上，要求文件元数据也要写入磁盘后，才能返回。

再比如，在访问管道或者网络套接字时，设置了 `O_ASYNC` 选项后，相应的 I/O 就是异步 I/O。这样，内核会再通过 `SIGIO` 或者 `SIGPOLL`，来通知进程文件是否可读写。

你可能发现了，这里的好多概念也经常出现在网络编程中。比如非阻塞 I/O，通常会跟 `select/poll` 配合，用在网络套接字的 I/O 中。

你也应该可以理解，“Linux 一切皆文件”的深刻含义。无论是普通文件和块设备、还是网络套接字和管道等，它们都通过统一的 VFS 接口来访问。

性能观测

学了这么多文件系统的原理，你估计也是迫不及待想上手，观察一下文件系统的性能情况了。

接下来，打开一个终端，SSH 登录到服务器上，然后跟我一起来探索，如何观测文件系统的性能。

容量

对文件系统来说，最常见的一个问题就是空间不足。当然，你可能本身就知道，用 `df` 命令，就能查看文件系统的磁盘空间使用情况。比如：

```
1 $ df /dev/sda1
2 Filesystem      1K-blocks    Used Available Use% Mounted on
3 /dev/sda1        30308240 3167020  27124836  11% /
```

 复制代码

你可以看到，我的根文件系统只使用了 11% 的空间。这里还要注意，总空间用 1K-blocks 的数量来表示，你可以给 `df` 加上 `-h` 选项，以获得更好的可读性：

```
1 $ df -h /dev/sda1
2 Filesystem      Size  Used Avail Use% Mounted on
3 /dev/sda1       29G   3.1G   26G   11% /
```

 复制代码

不过有时候，明明你碰到了空间不足的问题，可是用 `df` 查看磁盘空间后，却发现剩余空间还有很多。这是怎么回事呢？

不知道你还记不记得，刚才我强调的一个细节。除了文件数据，索引节点也占用磁盘空间。你可以给 `df` 命令加上 `-i` 参数，查看索引节点的使用情况，如下所示：

```
1 $ df -i /dev/sda1
2 Filesystem      Inodes   IUsed   IFree IUse% Mounted on
3 /dev/sda1       3870720 157460 3713260    5% /
```

 复制代码

索引节点的容量，（也就是 Inode 个数）是在格式化磁盘时设定好的，一般由格式化工具自动生成。当你发现索引节点空间不足，但磁盘空间充足时，很可能就是过多小文件导致的。

所以，一般来说，删除这些小文件，或者把它们移动到索引节点充足的其他磁盘中，就可以解决这个问题。

缓存

在前面 Cache 案例中，我已经介绍过，可以用 `free` 或 `vmstat`，来观察页缓存的大小。复习一下，`free` 输出的 Cache，是页缓存和可回收 Slab 缓存的和，你可以从 `/proc/meminfo`，直接得到它们的大小：

```
1 $ cat /proc/meminfo | grep -E "SReclaimable|Cached"
2 Cached:                748316 kB
3 SwapCached:              0 kB
4 SReclaimable:          179508 kB
```

 复制代码

话说回来，文件系统上的目录项和索引节点缓存，又该如何观察呢？

实际上，内核使用 Slab 机制，管理目录项和索引节点的缓存。/proc/meminfo 只给出了 Slab 的整体大小，具体到每一种 Slab 缓存，还要查看 /proc/slabinfo 这个文件。

比如，运行下面的命令，你就可以得到，所有目录项和各种文件系统索引节点的缓存情况：

复制代码

```
1 $ cat /proc/slabinfo | grep -E '^#|dentry|inode'
2 # name                <active_objs> <num_objs> <objsize> <objperslab> <pagesperslab> : tunables <lin
3 xfs_inode              0          0    960    17     4 : tunables    0      0      0 : slabdata    0
4 ...
5 ext4_inode_cache      32104   34590   1088    15     4 : tunables    0      0      0 : slabdata   2306  2
6 sock_inode_cache      1190     1242    704    23     4 : tunables    0      0      0 : slabdata    54
7 shmem_inode_cache     1622     2139    712    23     4 : tunables    0      0      0 : slabdata    93
8 proc_inode_cache      3560     4080    680    12     2 : tunables    0      0      0 : slabdata   340
9 inode_cache           25172   25818    608    13     2 : tunables    0      0      0 : slabdata   1986  1
10 dentry                76050  121296    192    21     1 : tunables    0      0      0 : slabdata   5776  5
```

这个界面中，dentry 行表示目录项缓存，inode_cache 行，表示 VFS 索引节点缓存，其余的则是各种文件系统的索引节点缓存。

/proc/slabinfo 的列比较多，具体含义你可以查询 man slabinfo。在实际性能分析中，我们更常使用 slabtop，来找到占用内存最多的缓存类型。

比如，下面就是我运行 slabtop 得到的结果：

复制代码

```
1 # 按下 c 按照缓存大小排序，按下 a 按照活跃对象数排序
2 $ slabtop
3 Active / Total Objects (% used)    : 277970 / 358914 (77.4%)
4 Active / Total Slabs (% used)      : 12414 / 12414 (100.0%)
5 Active / Total Caches (% used)     : 83 / 135 (61.5%)
6 Active / Total Size (% used)       : 57816.88K / 73307.70K (78.9%)
7 Minimum / Average / Maximum Object : 0.01K / 0.20K / 22.88K
8
9  OBJS ACTIVE  USE OBJ SIZE  SLABS OBJ/SLAB CACHE SIZE NAME
10 69804 23094  0%   0.19K   3324    21   13296K dentry
11 16380 15854  0%   0.59K   1260    13   10080K inode_cache
12 58260 55397  0%   0.13K   1942    30    7768K kernfs_node_cache
13   485   413  0%   5.69K    97     5    3104K task_struct
14  1472  1397  0%   2.00K    92    16    2944K kmalloc-2048
```

从这个结果你可以看到，在我的系统中，目录项和索引节点占用了最多的 Slab 缓存。不过它们占用的内存其实并不大，加起来也只有 23MB 左右。

小结

今天，我带你梳理了 Linux 文件系统的工作原理。

文件系统，是对存储设备上的文件，进行组织管理的一种机制。为了支持各类不同的文件系统，Linux 在各种文件系统实现上，抽象了一层虚拟文件系统（VFS）。

VFS 定义了一组所有文件系统都支持的数据结构和标准接口。这样，用户进程和内核中的其他子系统，就只需要跟 VFS 提供的统一接口进行交互。

为了降低慢速磁盘对性能的影响，文件系统又通过页缓存、目录项缓存以及索引节点缓存，缓和磁盘延迟对应用程序的影响。

在性能观测方面，今天主要讲了容量和缓存的指标。下一节，我们将会学习 Linux 磁盘 I/O 的工作原理，并掌握磁盘 I/O 的性能观测方法。

思考

最后，给你留一个思考题。在实际工作中，我们经常会根据文件名字，查找它所在路径，比如：

```
1 $ find / -name file-name
```

 复制代码

今天的问题就是，这个命令，会不会导致系统的缓存升高呢？如果有影响，又会导致哪种类型的缓存升高呢？你可以结合今天内容，自己先去操作和分析，看看观察到的结果跟你分析的是否一样。

欢迎在留言区和我讨论，也欢迎把这篇文章分享给你的同事、朋友。我们一起在实战中演练，在交流中进步。

Linux 性能优化实战

10 分钟帮你找到系统瓶颈

倪朋飞

微软资深工程师
Kubernetes 项目维护者



新版升级：点击「👤请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

©版权归极客邦科技所有，未经许可不得转载

上一篇 22 | Linux 性能优化答疑（三）

下一篇 24 | 基础篇：Linux 磁盘I/O是怎么工作的（上）

写留言

精选留言



石维康

👍 8

阻塞 I/O 和非阻塞 I/O的概念和同步和异步 I/O的区别是什么？

2019-01-11



白华

👍 6

课后题：我找了一个目录下的文件，用的这个命令`find / -type f -name copyright`然后slab top观察，发现dentry的SLABS和SIZE有了明显的提高，所以引起了目录项缓存的升高。在开始的时候dentry有一定的大小，我认为是缓存了/目录下系统基本的目录，但是系统后面下载、创建的内容是没有缓存的，使用查找命令会把这些都查找到然后缓存起来，所以使用find查找大量内容时候会造成性能下降。

前面看老男孩视频时候了解了inode和block。inode存储这些数据属性信息的，包含不限于文件大小、文件类型、文件权限、拥有者、硬链接数、所属组、修改时间，还包含指向文件实体的指针功能（inode节点---block的对应关系），但是inode唯独不包含文件名。文件名

不在inode里，在上级目录的block里；Block来存储实际数据用的，例如照片、视频等普通文件数据。

今天看到了dentry，定义是用来记录文件的名称、索引节点指针以及与其他目录项的关联关系。但是和老男孩老师讲的有所区别，希望老师帮我解惑

2019-01-11



伟忠

👍 2

机器上 df 查看占用了 200G，但 du 查看发现只有 90G，看网上的办法用 lsof | grep delete 查看，但没有找到，请问老师，这个可能是什么原因呢？

2019-01-12



DJH

👍 2

请教三个问题。

1. 目录项是维护在内核中的一个内存数据结构，包括文件名。

我的问题是：文件名不是也应该存储在磁盘上么？不可能仅仅存在于内存吧？

2. 缓冲 I/O，是指利用标准库缓存来减速文件的访问...

我的问题时：减速文件访问的原因是什么？

3. 阻塞/非阻塞与同步/非同步的区别是什么？

2019-01-11

作者回复

1. 目录项是表示目录之间的树状关系，而文件名则会存储到数据部分。

2. 不好意思，是个笔误，当然是加速。谢谢指出

3. 这个在文章中有简单的介绍，回来在答疑篇中再展开一些

2019-01-14



coyang

👍 2

课后题：

这个命令，会不会导致系统的缓存升高呢？

--> 会的

如果有影响，又会导致哪种类型的缓存升高呢？

--> /xfs_inode/ proc_inode_cache/dentry/inode_cache

实验步骤：

1. 清空缓存：echo 3 > /proc/sys/vm/drop_caches ; sync

2. 执行find：find / -name test

3. 发现更新top 4 项是：

OBJS ACTIVE USE OBJ SIZE SLABS OBJ/SLAB CACHE SIZE NAME

37400 37400 100% 0.94K 2200 17 35200K xfs_inode

36588 36113 98% 0.64K 3049 12 24392K proc_inode_cache

104979 104979 100% 0.19K 4999 21 19996K dentry
18057 18057 100% 0.58K 1389 13 11112K inode_cache

find / -name 这个命令是全盘扫描（既包括内存文件系统又包含本地的xfs【我的环境没有mount 网络文件系统】），所以 inode cache & dentry & proc inode cache 会升高。

另外，执行过了一次后再次执行find 就机会没有变化了，执行速度也快了很多，也就是下次的find大部分是依赖cache的结果。

2019-01-11



圣诞使者

👍 2

课后题，我感觉不会，应该只有用了目录的执行(x)权限内核才会缓存dentry，find只是用了目录的读(r)权限。

2019-01-11



xzyeah

👍 2

老师，我的理解，不会引起内存升高，因为文件名存在于目录项，目录项本身就存在于内存缓存。

2019-01-11



划时代

👍 1

执行find / -name file-name, vmstat 1观察, in/cs/us/sy项参数有上涨, cache项有一点上涨, free项有一点下降, 可想而知操作系统不会将所有目录项都存放在内存中, 再执行tree /时尤为明显。

回复老师问题, “内存打满”已处理属于某环境内存泄漏所致; Pss为0kB而Rss不为0kB, 在Ubuntu中较少见, bash进程有出现一个, 而在centos中则较多。

2019-01-11



Orcsir

👍 1

Flag。

关于思考题：

1.如果系统执行过一些操作，其中涉及到了要查找的file-name，那么在执行find命令的时候，应该是不会导致系统缓存的升高。

2.如果不是上述情况，或者执行过命令 echo 3 > /proc/sys/vm/drop_cache 操作的话，那么是会导致系统缓存升高的。只是通过vmstat观察的话，buffer与cache都用相应的增加，但是buffer的增加更明显。至于原因的话，推测是这样的：buffer增加是因为系统会缓存读取到的块设备中的数据。cache增加是因为内核创建了目录项缓存用来保存inode节点的相关信息，而目录项缓存属于cache部分，所以会导致cache也增加。但是为什么buffer会比cache增加的明显，暂时还没想明白。

2019-01-11



ninuxer

👍 1

打卡day24

阻塞非阻塞，同步异步再次mark下：

根据应用程序是否阻塞自身线程的运行，可以把文件 I/O 分为阻塞 I/O 和非阻塞 I/O；

根据是否等待响应结果，可以把文件 I/O 分为同步和异步 I/O

2019-01-11



渡渡鸟_linux

0

先清除页缓存，inode,dentry缓存：echo 3 > /proc/sys/vm/drop_caches

使用 find /etc/ -name "passwd" 未发现cache明显变化，执行速度非常快，因此考虑到是文件数量太少

使用 echo 3 > /proc/sys/vm/drop_caches ; find / -name "passwd" 后发现：

1. buffer 增加较少，40k 左右，
2. cache 增加较少，主要时 slab 增加较多，其中dentry,inode_cache,xfs_inode,proc_inode_cache 变化明显。
3. 可得到基本结论（不确定是否正确）： find 查找文件时，主要检索文件inode和dentry信息，对于逻辑块基本不检索(可能只是查找了存放文件名的数据块)

2019-01-20



苏宁

0

老师，能否推荐关于linux文件系统的书

2019-01-19



成为祝福

0

老师好，请问在slabtop中的inode_cache和ext4_inode_cache有什么区别呢？如果每个文件系统都有inode_cache，整个vfs的有效命名空间都映射到了对应的文件系统，vfs为什么还需要inode_cache呢？

2019-01-16



Something

0

为什么能把内存也能搞成文件系统？它又不是IO设备？

2019-01-15



曾刚

0

请问老师，后期有关于网络方面性能分析吗？多谢

2019-01-13

作者回复

有的

2019-01-14



xfan

0

我觉得目录项应该也是要交换到磁盘里，在内存里的话怎么持久化呢

2019-01-13

作者回复

目录项只是表示树状结构，用的时候动态创建就可以的，持久化还是要靠inode和文件本身的数据

2019-01-14



我来也

👍 0

[D23打卡]

以前就遇到过 df -h显示还有很多空间，可就是无法创建文件了。

后来才发现是index node节点用完了。

自此以后，我在每日服务器状态数据汇报中就添加了 df -i的统计数据。

“linux一切皆文件” 这设计太牛了。

今天通过老师的几个图，也更深入理解了应用程序到物理磁盘经过的过程。系统调用-vfs-文件系统-buffer-磁盘/网络。

2019-01-13



橘子

👍 0

操作系统中只能用一种文件系统还是可以多个文件系统在同一个操作系统中共存？共存的话如何查看操作系统用了哪些文件系统，这些文件系统由哪些设备使用，比如磁盘用的什么文件系统，网络用的什么文件系统，内存用的什么文件系统

2019-01-13

作者回复

多种文件系统可以同时使用，可以根据挂载信息查看。

比如 /proc/mounts

2019-01-14



大甜菜

👍 0

find应该就是遍历目录，查找目录下面是否有所查找的文件

目录本身也是文件，他储存的内容就是文件名和inode号的映射，既然读过文件，缓存肯定回事升高

2019-01-12



未来小鬼

👍 0

同步，异步，阻塞非阻塞。。。从没搞懂。。。

2019-01-12